# On the Computational Complexity of MapReduce*

Benjamin Fish, Jeremy Kun, Ádám D. Lelkes, Lev Reyzin and György Turán

Department of Mathematics, Statistics, and Computer Science
University of Illinois at Chicago
Chicago, IL 60607
{bfish3,jkun2,alelke2,lreyzin,gyt}@math.uic.edu

**Abstract**

In this paper we study the complexity class MRC, introduced by Karloff et al. [6], whose aim was to formally capture MapReduce computations. First, we observe that the original definition of Karloff et al. allows for non-uniform computation, and we introduce a uniform model of MRC to more accurately capture real-world uses of MapReduce. Then, we initiate a study of uniform MRC from a computational complexity standpoint. We show that regular languages, and moreover all of sublogarithmic space, lies in constant round MRC. We also give a lower bound showing that allowing more rounds of MapReduce computation cannot generically simulate all computations done with a smaller number of rounds.

*This paper is eligible for a best student paper award. Benjamin Fish, Jeremy Kun, and Ádám D. Lelkes are full-time students who contributed significantly to the paper.

# 1 Introduction

MapReduce is a programming model originally developed to separate algorithm design from the engineering challenges of massively distributed computing. A programmer can separately implement a "map" function and a "reduce" function that satisfy certain constraints, and the underlying MapReduce technology handles all the communication, load balancing, fault tolerance, and scaling. MapReduce frameworks and their variants have been successfully deployed in industry by Google [2], Yahoo! [9], and many others.

In 2010 Karloff et al. [6] initiated a principled study of MapReduce as a complexity class, providing the crucial definitions and comparing it with the classical PRAM models of parallel computing. But to our knowledge, since this initial paper, most of the work on MapReduce has focused on algorithmic issues.

In this paper we continue the work of [6] by laying a more precise theoretical foundation for studying MapReduce computations. In particular, we observe that the original definition for MapReduce is non-uniform, allowing the complexity class to contain undecideable languages (Section 3.2). We refine the definition of [6] to make a uniform model and to more finely track the parameters involved (Section 3.3). We then prove two main theorems for our uniform model: a space upper bound, that SPACE($o(\log n)$) has constant-round MapReduce computations (Section 4.2), and a time lower-bound within MapReduce which informally states that additional rounds cannot compensate for additional time (Section 5). We conclude with a collection of open questions raised by our work (Section 6).

# 2 Background and previous work

The MapReduce protocol can be roughly described as follows. The input data is given as a list of key-value pairs, and over a series of rounds two things happen per round: a "mapper" is applied to each key-value pair independently (in parallel), and then for each distinct key a "reducer" is applied to all corresponding values. As part of its duties, the underlying MapReduce technology figures out how many mappers and reducers to run in each round in parallel, and performs the key grouping automatically.

The canonical example is counting word frequencies with a two-round MapReduce protocol. The inputs are (index, word) pairs, the first mapper maps $(k, v) \mapsto (v, k)$, and the first reducer computes the sum of the word frequencies for the given key. In the second round the mapper sends all data to a single processor via $(k, n_k) \mapsto (1, (k, n_k))$, and the second processor formats the output appropriately.

The emphasis in MapReduce is on processing massive data sets. So MapReduce programs work best when every reducer only works on a substantially sublinear portion of the input, and the strict modularization prohibits reducers from communicating within a round. All communication happens across rounds through mappers, which are severely limited in power by the independence requirement. Finally, it's understood in practice that the critical quantity to optimize for is the number of rounds, so algorithms which cannot avoid a large number of rounds are considered inefficient and unsuitable for MapReduce.

The influential paper of Karloff et al. was the first to view MapReduce as a complexity class and to codify the theoretical notion of an efficient MapReduce algorithm. In [6] they compare MapReduce to standard classes such as NC. In particular, they exhibit the power of interleaving

parallel and sequential computation by giving a two round MapReduce protocol for computing minimum spanning trees in dense graphs, contrasting it with the $\Omega(\log(n))$-round lower bound for NC. They further prove a theorem that MapReduce algorithms can simulate CREW PRAMs which use subquadratic total memory and processors. Karloff et al. name their class $\text{MRC}^i$ (MRC stands for "Map Reduce Class;" the $i$ denotes $O(\log^i(n))$ rounds).

Since [6], there has been extensive work in developing efficient algorithms in the MapReduce framework. For example, Moseley et al. analyzed a sampling technique allowing them to translate sequential greedy algorithms into $\text{MRC}^1$ algorithms with a small loss of quality [7]. Farahat et al. [3] investigate the potential for sparsifying distributed data using random projections. Kamara and Raykova [5] develop a homomorphic encryption scheme for MapReduce. And much work has been done on graph problems such as connectivity, matchings, sorting and searching [4]. Finally, Chu et al. [1] demonstrate the potential to express any statistical-query learning algorithm in the MapReduce framework (though [1] came before [6], and hence does not explicitly use their model).

The study of MapReduce has resulted in a wealth of new and novel algorithms. As such, a more detailed study of the power of MapReduce is warranted. Our paper contributes to this by establishing a more precise definition of the MapReduce complexity class, proving upper bounds for deterministic space, and a lower bound within MapReduce.

# 3 Definition of MRC

## 3.1 Mappers, reducers, and the MRC protocol

The central piece of data in MRC is the key-value pair. We denote a key-value pair by a pair of strings $\langle k, v\rangle$, where $k$ is the key and $v$ is the value. An input to an MRC machine is a list of key-value pairs $\langle k_i, v_i\rangle_{i=1}^N$ and we denote the total size of the input by $n = \sum_{i=1}^N |k_i| + |v_i|$.

**Definition 1.** A *mapper* $\mu$ is a Turing machine which accepts as input a single key-value pair $\langle k, v\rangle$ and produces a list of key-value pairs $\langle k'_1, v'_1\rangle, \ldots, \langle k'_s, v'_s\rangle$.

Note in particular that a mapper may change the key-value pair representation, but its choices are independent across multiple inputs.

**Definition 2.** A *reducer* $\rho$ is a Turing machine which accepts as input a key $k$ and a list of values $\langle v_1, \ldots, v_m\rangle$, and produces as output the same key and a new list of values $\langle v'_1, \ldots, v'_M\rangle$.

**Definition 3.** For a decision problem, an input string $x \in \{0,1\}^*$ to an MRC machine is the list of pairs $\langle i, x_i\rangle_{i=1}^n$ describing the index and value of each bit. We will denote by $\langle x\rangle$ the list $\langle i, x_i\rangle$.

*Remark* 1. Because our space constraints will be $O(n^c)$ for some $c < 1$, this blowup does not constrain the class of solvable problems.

An MRC machine operates in rounds. In each round, a set of mappers running in parallel first process all the key-value pairs. Then the pairs are partitioned (by a mechanism called "shuffle and sort" that is not considered part of the runtime of an MRC machine) so that each reducer only receives key-value pairs for a single key. Then the reducers process their pieces of the data in parallel, and the results are merged to form the list of key-value pairs for the next round. More formally:

**Definition 4.** An *R-round MRC machine* is an alternating list of mappers and reducers $M = (\mu_1, \rho_1, \ldots, \mu_R, \rho_R)$. The execution of the machine is as follows.

For each $r = 1, \ldots, R$:

1. Let $U_{r-1} = (\langle k_{1,r-1}, v_{1,r-1}\rangle, \ldots, \langle k_{N_{r-1},r-1}, v_{N_{r-1},r-1}\rangle)$ be the list of key-value pairs processed from the last round (or the input pairs when $r = 1$). Apply $\mu_r$ to each key-value pair of $U_{r-1}$ to get the multiset
$$V_r = \bigcup_{\langle k,v\rangle \in U_{r-1}} \mu_r(k, v).$$
2. Shuffle-and-sort groups the values by key. Call each piece
$$V_{k,r} = \{k, (v_{k,1}, \ldots, v_{k,s_k})\}.$$
3. Assign a different copy of reducer $\rho_r$ to each $V_{k,r}$ (run in parallel) and set $U_r = \bigcup_k \rho_r(V_{k,r})$.

The output is the final set of key-value pairs. For decision problems, we define $M$ to accept $\langle x\rangle$ if in the final round $U_R$ is the empty set. We say $M$ *decides* a language $L$ if it accepts $\langle x\rangle$ if and only if $x \in L$.

The central caveat that makes MRC an interesting class is that the reducers have space constraints that are sublinear in the size of the input string. In other words, no sequential computation may happen that has random access to the entire input. Cooperation between reducers is obtained not by message passing or shared memory, but rather across rounds. This is both the central difficulty in designing MRC algorithms, and the key to efficient and fault-tolerant implementations in practice.

Before we give the main definition of [6], it is worth noting that the original definitions were for PRAMs. However, because we wish to relate MapReduce to classical complexity classes like P and L, it is more natural to reformulate the definitions here in terms of Turing machines.

### 3.2 The definition of Karloff et al. and nonuniformity

We can now describe what it means for a decision problem to be in a bounded-round MRC complexity class. This is a less general form of the definition originally given by Karloff et al. [6], and we refer to it as Nonuniform Deterministic MRC.

**Definition 5** (Nonuniform Deterministic MRC (Karloff-Suri-Vassilvitskii))**.** A language $L$ is in *nonuniform MRC*$[f(n), g(n)]$ if there is a constant $0 < c < 1$ and a sequence of mappers and reducers $\mu_1, \rho_1, \mu_2, \rho_2, \ldots$ such that for all $x \in \{0,1\}^n$ the following is satisfied:

1. Letting $R = O(f(n))$ and $M = (\mu_1, \rho_1, \ldots, \mu_R, \rho_R)$, $M$ accepts $x$ if and only if $x \in L$.
2. For all $1 \le r \le R$, $\mu_r, \rho_r$ use $O(n^c)$ space and $O(g(n))$ time.
3. Each $\mu_r$ outputs $O(n^c)$ distinct keys in round $r$.

*Remark* 2. The last item in the list ensures that there are at most $O(n^c) = o(n)$ processors running in any round. Furthermore, this imposes a total running time bound of $O(nf(n)g(n))$ on an MRC machine. Here "total" means the sum of the running times of all parallel mappers and reducers over all rounds. See Remark 4 and Section 5 for a related discussion of time-bounded Turing machines simulating MRC machines.

The original definition of [6] required a polylogarithmic number of rounds, and also allowed completely different MapReduce machines for different input sizes. It is not hard to see that this model is less powerful, but it contains undecidable problems nonetheless. Indeed, nonuniform MRC$[n, \sqrt{n}]$ accepts all unary langauges, i.e. languages of the form $L \subseteq \{1^n \mid n \in \mathbb{N}\}$.

**Lemma 1.** *Let $L$ be a unary language. Then $L$ is in nonuniform* MRC$[n, \sqrt{n}]$.

*Proof.* We define the mappers and reducers as follows. Let $\mu_1$ distribute the input as contiguous blocks of $\sqrt{n}$ bits, $\rho_1$ compute the length of its input, $\mu_2$ send the counts to a single processor, and $\rho_2$ add up the counts, i.e. find $n = |x|$ where $x$ is the input. At this point the input data has been reduced to a single key-value pair $\langle \star, n \rangle$. Then let $\rho_i$ for $i \geq 3$ be the reducer that on input $\langle \star, i-3 \rangle$ accepts if and only if $1^{i-3} \in L$ and otherwise outputs the input. Let $\mu_i$ for $i \geq 3$ send the input to a single processor. Then $\rho_{n+3}$ will accept if and only if $x$ is in $L$. Note that $\rho_1, \rho_2$ take $O(\sqrt{n})$ time, and all other mappers and reducers take constant time. Further note that all mappers and reducers are in SPACE$(\sqrt{n})$. So $L$ is in non-uniform MRC$[n, \sqrt{n}]$. □

In particular, Lemma 1 implies that nonuniform MRC$[n, \sqrt{n}]$ contains the unary version of the Halting problem. A more careful analysis shows all unary languages are in MRC$[\log n, \sqrt{n}]$, by having $\rho_{i+3}$ check $2^i$ strings for membership in $L$.

### 3.3 Uniform MRC

In the interest of relating MRC to uniform classes like P and L, we present a uniform version of the definition in the previous section. In particular, we require that every mapper and reducer arise as separate runs of the same Turing machine $M$. Our Turing machine $M(m, r, n, y)$ will accept as input the current round number $r$, a bit $m$ denoting whether to run the $r$-th map or reduce function, the total number of rounds $n$, and the corresponding input $y$. The reason for including $r, n$ is to allow a mapper to do things like "Distribute the keys in equal-sized blocks among $\sqrt{n}$ processors," and vary their behavior for the first and last rounds.[1]

**Definition 6** (Uniform Deterministic MRC)**.** A language $L$ is said to be in MRC$[f(n), g(n)]$ if there is a constant $0 < c < 1$, an $O(n^c)$-space and $O(g(n))$-time Turing machine $M(m, r, n, y)$, and an $R = O(f(n))$, such that for all $x \in \{0,1\}^n$, the following holds.

1. Letting $\mu_r = M(1, r, n, -), \rho_r = M(0, r, n, -)$, the MRC machine $M_R = (\mu_1, \rho_1, \ldots, \mu_R, \rho_R)$ accepts $x$ if and only if $x \in L$.
2. Each $\mu_r$ outputs $O(n^c)$ distinct keys.

*Remark* 3. By $M(1, r, n, -)$, we mean that the tape of $M$ is initialized by the string $\langle 1, r, n \rangle$. In particular, this prohibits an MRC algorithm from having an exponential number of rounds; the space constraints would prohibit it from storing the round number.

*Remark* 4. Note that a polynomial time Turing machine with sufficient time can trivially simulate a uniform MRC machine. All that is required is for the machine to perform the key grouping manually, and run the MRC machine as a subroutine. As such, MRC$[\text{poly}(n), \text{poly}(n)] \subset P$. We give a more precise computation of the amount of overhead required in the proof of Theorem 5.

[1] Karloff et al.'s original definition does not give the mappers and reducers access to $n$. While they implicitly accounted for it by nonuniformity, we explicitly add it here. Indeed, in practice most MapReduce implementations include the the implicit ability to access the number of redcuers, which is in turn determined by the size of the input.

We further define *randomized* MRC$[f(n), g(n)]$ to be the natural extension of MRC$[f(n), g(n)]$ that provides $M$ access to random bits and requires the answer is correct with probability bounded away from $1/2$.

**Definition 7.** Define by MRC$^i$ the union of uniform MRC classes

$$\text{MRC}^i = \bigcup_{k \in \mathbb{N}} \text{MRC}[\log^i(n), n^k].$$

So in particular $\text{MRC}^0 = \bigcup_{k \in \mathbb{N}} \text{MRC}[1, n^k]$. We will work exclusively with the uniform model the remainder of the paper.

# 4 Upper bounds

## 4.1 Regular langauges are in MRC$^0$

In this section we prove that the class REGULAR of regular languages is in MRC$^0$. It is well known that SPACE$(O(1))$ = REGULAR [8], and so this result can be viewed as a warm-up to the next section in which we prove SPACE$(o(\log n)) \subset$ MRC$^0$. Indeed, both proofs share the same flavor, which we sketch before proceeding to the details.

In the first round each parallel processor receives a contiguous portion of the input string and constructs a state transition function using the data of the globally known DFA. Though only the processor with the beginning of the string knows the true state of the machine during its portion of the input, all processors can still compute the *entire* table of state-to-state transitions for the given portion of input. In the second round, one processor collects the transition tables and chains together the computations, and this step requires only the first bit of input and the list of tables.

We can count up the space and time requirements to prove the following theorem.

**Theorem 2.** REGULAR $\subsetneq$ MRC$^0$

*Proof.* Let $L$ be a regular language and $D$ a deterministic finite automaton recognizing $L$. Define the first mapper so that the $j^{\text{th}}$ processor has the bits from $j\sqrt{n}$ to $(j+1)\sqrt{n}$. This means we have $K = O(\sqrt{n})$ processors in the first round. Because the description of $D$ is independent of the size of the input string, we also assume each processor has access to the relevant set of states $S$ and the transition function $t : S \times \{0, 1\} \to S$.

We now define $\rho_1$. Fix a processor $j$ and call its portion of the input $y$. The processor constructs a table $T_j$ of size at most $|S|^2 = O(1)$ by simulating $D$ on $y$ starting from all possible states and recording the state at the end of the simulation. It then passes $T_j$ and the first bit of $y$ to the single processor in the second round.

In the second round the sole processor has $K$ tables $T_j$ and the first bit $x_1$ of the input string $x$ (among others but these are ignored). Treating $T_j$ as a function, this processor computes $q = T_K(\ldots T_2(T_1(x_1)))$ and accepts if and only if $q$ is an accepting state. This requires $O(\sqrt{n})$ space and time and proves containment. To show this is strict, inspect the prototypical problem of deciding whether the majority of bits in the input are 1's. □

### 4.2 Space $o(\log(n))$ is in MRC$^0$

We now move on to prove SPACE($o(\log n)$) $\subset$ MRC$^0$. It is worth noting that this is a strictly stronger statement than Theorem 2. That is, REGULAR = SPACE($O(1)$) $\subsetneq$ SPACE($o(\log n)$). Several non-trivial examples of languages that witness the strictness of this containment are given in [10].

The proof is very similar to the proof of Theorem 2: Instead of the processors computing the entire table of state-to-state transitions of a DFA, the processors now compute the entire table of all transitions possible among the configurations of the work tape of a Turing machine that uses $o(\log n)$ space.

**Theorem 3.** SPACE($o(\log n)$) $\subset$ MRC$^0$.

*Proof.* Let $L$ be a language in SPACE($o(\log n)$) and $T$ a Turing machine recognizing $L$ in polynomial time and $o(\log(n))$ space, with a read/write work tape $W$. Define the first mapper so that the $j^{\text{th}}$ processor has the bits from $j\sqrt{n}$ to $(j+1)\sqrt{n}$. Let $\mathcal{C}$ be the set of all possible configurations of $W$ and let $S$ be the states of $T$. Since the size of $S$ is independent of the input, we can assume that each processor has the transition function of $T$ stored on it.

Now we define $\rho_1$ as follows: Each processor $j$ constructs the graph of a function $T_j : \mathcal{C} \times \{L, R\} \times S \to \mathcal{C} \times \{L, R\} \times S$, which simulates $T$ when the read head starts on either the left or right side of the $j$th $\sqrt{n}$ bits of the input and $W$ is in some configuration from $\mathcal{C}$. It outputs whether the read head leaves the $y$ portion of the read tape on the left side, the right side, or else accepts or rejects. To compute the graph of $T_j$, processor $j$ simulates $T$ using its transition function, which takes polynomial time.

Next we show that the graph of $T_j$ can be stored on processor $j$ by showing it can be stored in $O(\sqrt{n})$ space. Since $W$ is by assumption size $o(\log n)$, each entry of the table is $o(\log n)$, so there are $2^{o(\log n)}$ possible configurations for the tape symbols. There are also $o(\log n)$ possible positions for the read/write head, and a constant number of states $T$ could be in. Hence $|\mathcal{C}| = 2^{o(\log n)} o(\log n) = o(n^{1/3})$. Then processor $j$ can store the graph of $T_j$ as a table of size $O(n^{1/3})$.

The second map function $\mu_2$ sends each $T_j$ (there are $\sqrt{n}$ of them) to a single processor. Each is size $O(n^{1/3})$, and there are $\sqrt{n}$ of them, so a single processor has space for all of the tables. Using these tables, the final reduce function can now simulate $T$ from starting state to either the accept or reject state by computing $q = T_k^*(\ldots T_2^*(T_1^*(\emptyset, L, \mathit{initial})))$ for some $k$, where $\emptyset$ denotes the initial configuration of $T$, *initial* is the initial state of $T$, and $q$ is either in the accept or reject state. Note $T_j^*$ is the modification of $T_j$ such that if $T_j(x)$ outputs $L$, then $T_j^*(x)$ outputs $R$ and vice versa. This is necessary because if the read head leaves the $j^{\text{th}}$ $\sqrt{n}$ bits to the right, it enters the $j+1^{\text{th}}$ $\sqrt{n}$ bits from the left, and vice versa. Finally, accept if and only if $q$ is in an accept state.

This algorithm successfully simulates $T$, which decides $L$, and only takes a constant number of rounds, so $L$ is in MRC$^0$. □

## 5 Time vs. rounds

In this section we prove the informal theorem that an MRC machine with a given number of rounds cannot capture all computations done with a smaller number of rounds but more time. In particular, Theorem 5 states that for any $f(n), g(n)$, there is a language $L \notin \text{MRC}[f(n), g(n)]$ that

can be decided by a *one round* MRC machine with $h(n) = \text{poly}(f(n), g(n))$ time. In addition, Theorem 5 allows us to separate subsets of MRC from P; see Corollary 6.

As a sketch of the proof, we observe that a Turing machine can simulate an MRC machine with measurably polynomial overhead, and then we pad a linear space (and polynomial time) language so that a single reducer can decide it. Our theorem relies on the complexity assumption that P $\neq$ PSPACE. Note that this is a weaker assumption than P $\neq$ NP.

First we prove a technical lemma that can be viewed as a "time hierarchy within $\text{TISP}(T(n), n)$," and may be of independent interest. The lemma will provide us with the language required for the main theorem. Recall that $\text{TISP}(T(n), S(n))$ is the set of languages decided by a Turing machine that on every input of length $n$ takes at most $O(T(n))$ steps and uses at most $O(S(n))$ cells of its tapes. Note that in general $\text{TISP}(T(n), S(n)) \neq \text{TIME}(T(n)) \cap \text{SPACE}(S(n))$.

**Lemma 4.** *Suppose* P $\neq$ PSPACE. *If $T$ and $T'$ are time-constructible functions satisfying*

$$n^2 T(n) = O(T'(n)) = O(n^k)$$

*for some $k$ then*

$$\text{TISP}(T'(n), n) \setminus \text{TIME}(T(n)) \neq \emptyset.$$

*Proof.* First we claim $\text{SPACE}(n) \setminus \text{TIME}(T(n))$ is nonempty. The language TQBF of true quantified Boolean formulas is in $\text{SPACE}(n)$ and PSPACE-complete, so $\text{SPACE}(n) \subseteq \text{TIME}(T(n)) \subseteq \text{P}$ would imply P = PSPACE. Let $L$ be a language in $\text{SPACE}(n) \setminus \text{TIME}(T(n))$. If $L \in \text{TISP}(T'(n), n)$ then we are done.

Otherwise let $c_n$ be the smallest integer such that a deterministic Turing machine can decide $L$ on input strings of length $n$ in time $n^{c_n}$ using linear space, let $b$ be the largest integer such that $T'(n) = \Omega(n^b)$ and let $a$ be the smallest integer such that $T(n) = O(n^a)$. Observe that $a < b$ since $n^2 T(n) = O(T'(n))$. Pad $L$ with $n^{c_n/b}$ ones and call this language $L'$, i.e. let

$$L' = \{x 1^{n^{c_n/b}} \mid x \in L, n = |x|\}.$$

Let $N = n + n^{c_n/b}$. Then $L' \in \text{TISP}(T'(N), N)$ since $T'(N) = \Omega(N^b) = \Omega(n^{c_n})$.

On the other hand, assume for contradiction that $L' \in \text{TIME}(T(N))$. Then it follows that $L \in \text{TIME}(T(n + n^{c_n/b})) \subseteq \text{TIME}((n + n^{c_n/b})^a)) = \text{TIME}(n^{ac_n/b})$. This contradicts the choice of $c_n$: since $a < b < c_n$ for infinitely many values of $n$, it follows that $\frac{a}{b} \leq \frac{c_n - 1}{c_n}$, hence $c_n \cdot \frac{a}{b} \leq c_n - 1$. Thus we can conclude that $L' \in \text{TISP}(T'(N), N) \setminus \text{TIME}(T(N))$.

□

*Remark* 5. Note that a stronger complexity conjecture (e.g., the Exponential Time Hypothesis) can be used to tighten the bound between $T(n)$ and $T'(n)$: assume $L$ requires $2^{\Theta(n)}$ time and pad with $2^{cn^{1/k}}$ ones.

**Theorem 5.** *Suppose* P $\neq$ PSPACE. *Let $f(n), g(n) = \text{poly}(n)$ be time-constructible functions and*

$$T(n) = nf(n)g(n) + n^2 \log(n) f(n).$$

*Also let $T'(n)$ be a time-constructible function such that*

$$\forall 0 < c < 1, n^2 T(n^{1+c}) = O(T'(n)).$$

*Then*

$$\text{MRC}[1, T'(n)] \not\subset \text{MRC}[f(n), g(n)].$$

*Proof.* In short, we use the previous lemma to obtain a sufficiently hard language and pad the input to fit on a single reducer. This language will be in $\text{MRC}[1, T'(n)]$ but not $\text{MRC}[f(n), g(n)]$.

We first show that any language in $\text{MRC}[f(n), g(n)]$ can be simulated in time $O(T(n))$, i.e. $\text{MRC}[f(n), g(n)] \subset \text{TIME}(T(n))$. The $r$-th round is simulated by applying $\mu_r$ to each key-value pair in sequence, shuffle-and-sorting the new key-value pairs, and then applying $\rho_r$ to each appropriate group of key-value pairs sequentially. Indeed, $M(m, r, n, -)$ can be simulated naturally by keeping track of $m$ and $r$, and adding $n$ to the tape at the beginning of the simulation. Each application of $\mu_r$ takes $O(g(n))$ time, for a total of $O(ng(n))$ time. Since each mapper outputs no more than $O(n^c) = O(n^2)$ keys, and each mapper and reducer is in $\text{SPACE}(O(n^c))$, there are no more than $O(n^2)$ keys to sort. Then shuffle-and-sorting takes $O(n^2 \log n)$ time, and the applications of $\rho_r$ also take $O(ng(n))$ time. So a round takes $O(ng(n) + n^2 \log n)$ time. Note that keeping track of $m$,$r$, and $n$ takes no more than the above time. So over $O(f(n))$ rounds, the simulation takes $O(nf(n)g(n) + n^2 \log(n) f(n)) = O(T(n))$ time.

Fix some arbitrary $0 < c < 1$, and let $L$ be a language which is in $\text{TISP}(T'(n), n)$ but not $\text{TIME}(T(n^{1+c}))$, which exists from the previous lemma. Pad $L$ with $n^{1+c}$ ones, and call this new language $L'$, i.e. let $L' = \{x1^{n^{1+c}} \mid x \in L, n = |x|\}$. Let $N = n + n^{1+c}$. There is an $\text{MRC}[1, T'(N)]$ algorithm to decide $L'$: the first mapper discards all the key-value pairs except those in the first $n$, and sends all remaining pairs to a single reducer. The space consumed by all pairs is $O(n) = O(N^{1/(1+c)})$. This reducer decides $L$, which is possible since $L \in \text{TISP}(T'(n), n)$. We now claim $L'$ is not in $\text{MRC}[f(N), g(N)]$. If it were, then $L'$ would be in $\text{TIME}(T(N))$. A Turing machine that decides $L'$ in $T(N)$ time can be modified to decide $L$ in $T(N)$ time: pad the input string with $n^{1+c}$ ones and use the decider for $L'$. This shows $L$ is in $\text{TIME}(T(n^{1+c}))$, a contradiction. □

As a concrete example of Theorem 5, $\text{MRC}[1, n^6] \not\subset \text{MRC}[\log(n), n]$.

**Corollary 6.** *If* $\text{L} \subset \text{MRC}[n^j, n^k]$ *for some* $j, k$*, then* $\text{L} \neq \text{P}$.

*Proof.* $\text{P} = \text{PSPACE}$ already implies $\text{L} \neq \text{P}$ by the space hierarchy theorem, so Theorem 5 provides a language in $\text{P} \setminus \text{L}$ in the case that $\text{P} \neq \text{PSPACE}$. □

It is readily possible that $\text{L} \not\subset \text{MRC}[n^j, n^k]$ for any $j, k$, but this would be a very interesting lower bound on the computational power of MRC.

## 6 Open Problems

In light of Corollary 6 and the fact that sublogarithmic space has constant-round MRC protocols, the central open problem is to determine where $L$ falls. In particular, is $L \subset \text{MRC}[n^j, n^k]$ for some $j, k$? We believe this not to be the case, although some canonical $L$ problems are. For example, graph connectivity is in $\text{MRC}[n, \text{poly}(n)]$ by the simple observation that matrix multiplication is in $\text{MRC}^0$. Graph connectivity in particular provides an interesting open problem. While dense graph connectivity is in randomized $\text{MRC}^0$, general graph connectivity is only known to be in $\text{MRC}^1$. We conjecture that graph connectivity has no constant round randomized MRC algorithm.

Second, Theorem 5 raises the question of how many rounds one must add to compensate for added time in a single round. Using the notation from the theorem, is there a larger $f'(n)$ such that $\text{MRC}[1, T'(n)] \subset \text{MRC}[f'(n), g(n)]$? If not, can one choose a $g'(n) = o(T'(n))$ that works?

Another major question is whether $\mathrm{MRC}[\mathrm{poly}(n), \mathrm{poly}(n)] \subsetneq P$? Any fixed parameters will admit a strict containment by the time hierarchy theorem, but this more general question can be phrased as, "Are there efficiently solvable problems, any algorithm for which requires random access to the entire string at every step?" We conjecture a positive answer.

## Acknowledgements

We thank Benjamin Moseley for helpful discussions.

## References

[1] Cheng-Tao Chu, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary R. Bradski, Andrew Y. Ng, and Kunle Olukotun. Map-reduce for machine learning on multicore. In *NIPS*, pages 281–288, 2006.

[2] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[3] Ahmed K. Farahat, Ahmed Elgohary, Ali Ghodsi, and Mohamed S. Kamel. Distributed column subset selection on mapreduce. In *ICDM*, pages 171–180, 2013.

[4] Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the mapreduce framework. In *Proceedings of the 22Nd International Conference on Algorithms and Computation*, ISAAC'11, pages 374–383, Berlin, Heidelberg, 2011. Springer-Verlag.

[5] Seny Kamara and Mariana Raykova. Parallel homomorphic encryption. In *Financial Cryptography Workshops*, pages 213–225, 2013.

[6] Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 938–948, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics.

[7] Ravi Kumar, Benjamin Moseley, Sergei Vassilvitskii, and Andrea Vattani. Fast greedy algorithms in mapreduce and streaming. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '13, pages 1–10, New York, NY, USA, 2013. ACM.

[8] J. C. Shepherdson. The reduction of two-way automata to one-way automata. *IBM J. Res. Dev.*, 3(2):198–200, April 1959.

[9] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In Mohammed G. Khatib, Xubin He, and Michael Factor, editors, *MSST*, pages 1–10. IEEE Computer Society, 2010.

[10] A. Szepietowski. *Turing Machines with Sublogarithmic Space*. Ernst Schering Research Foundation Workshops. Springer, 1994.